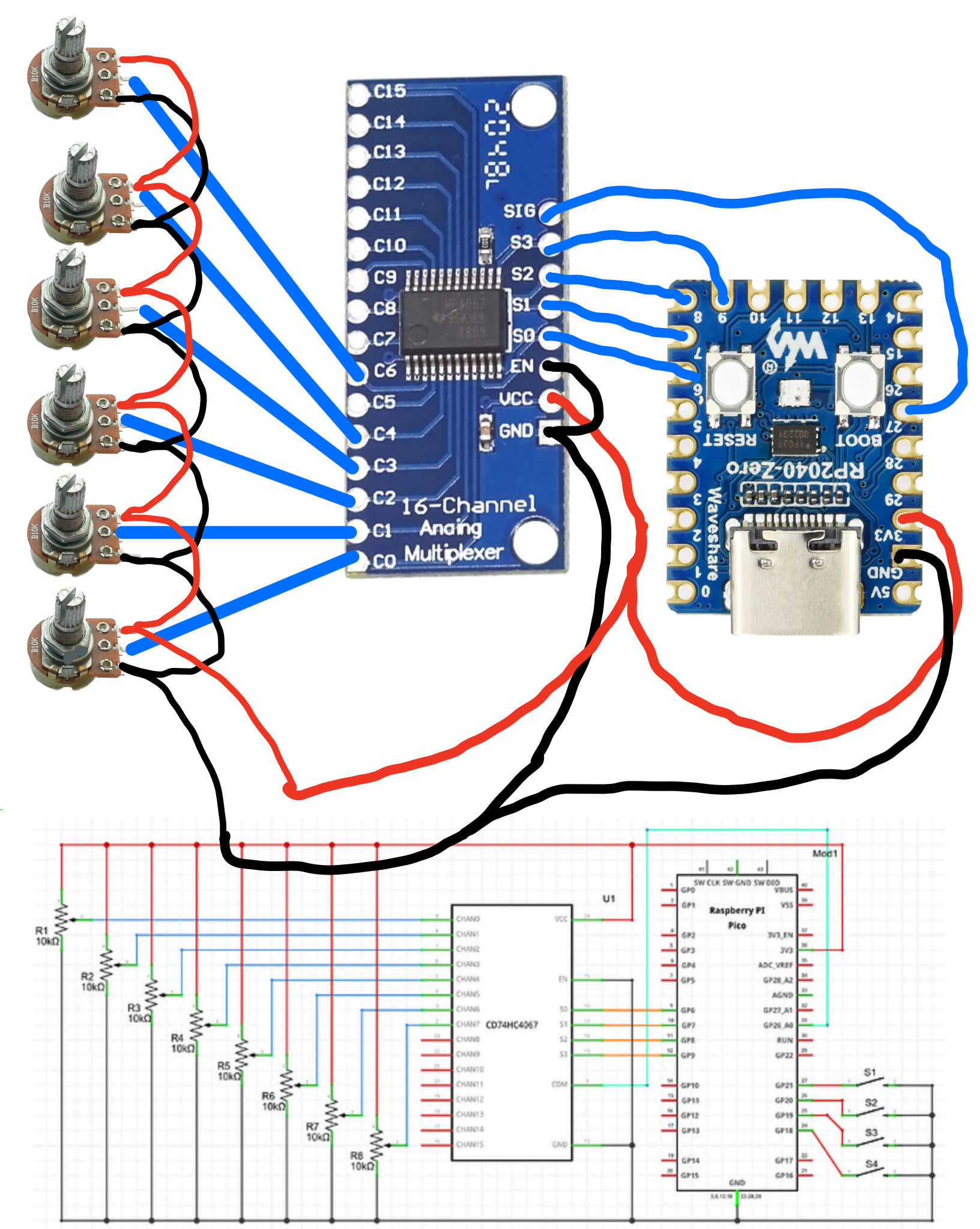
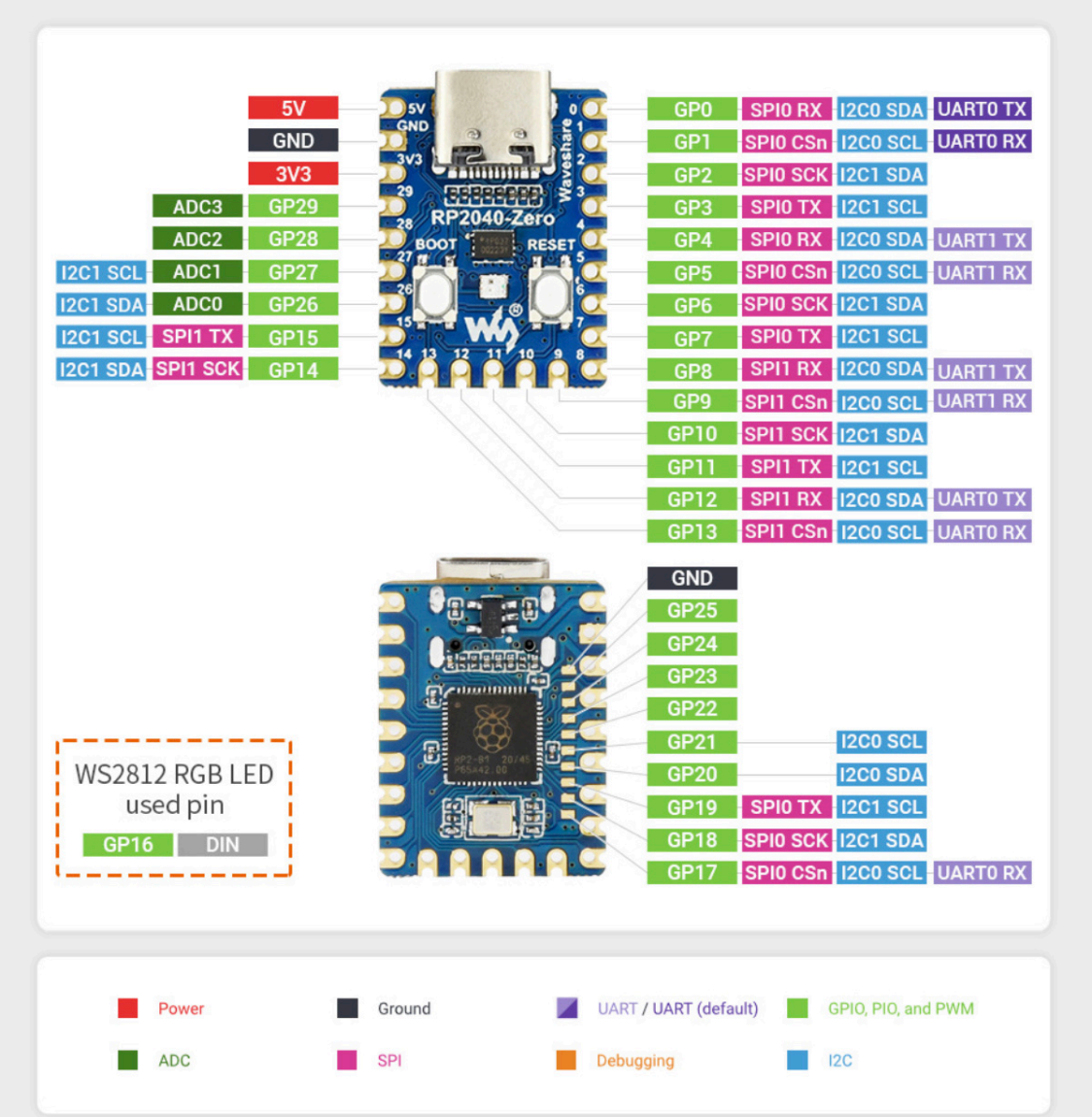
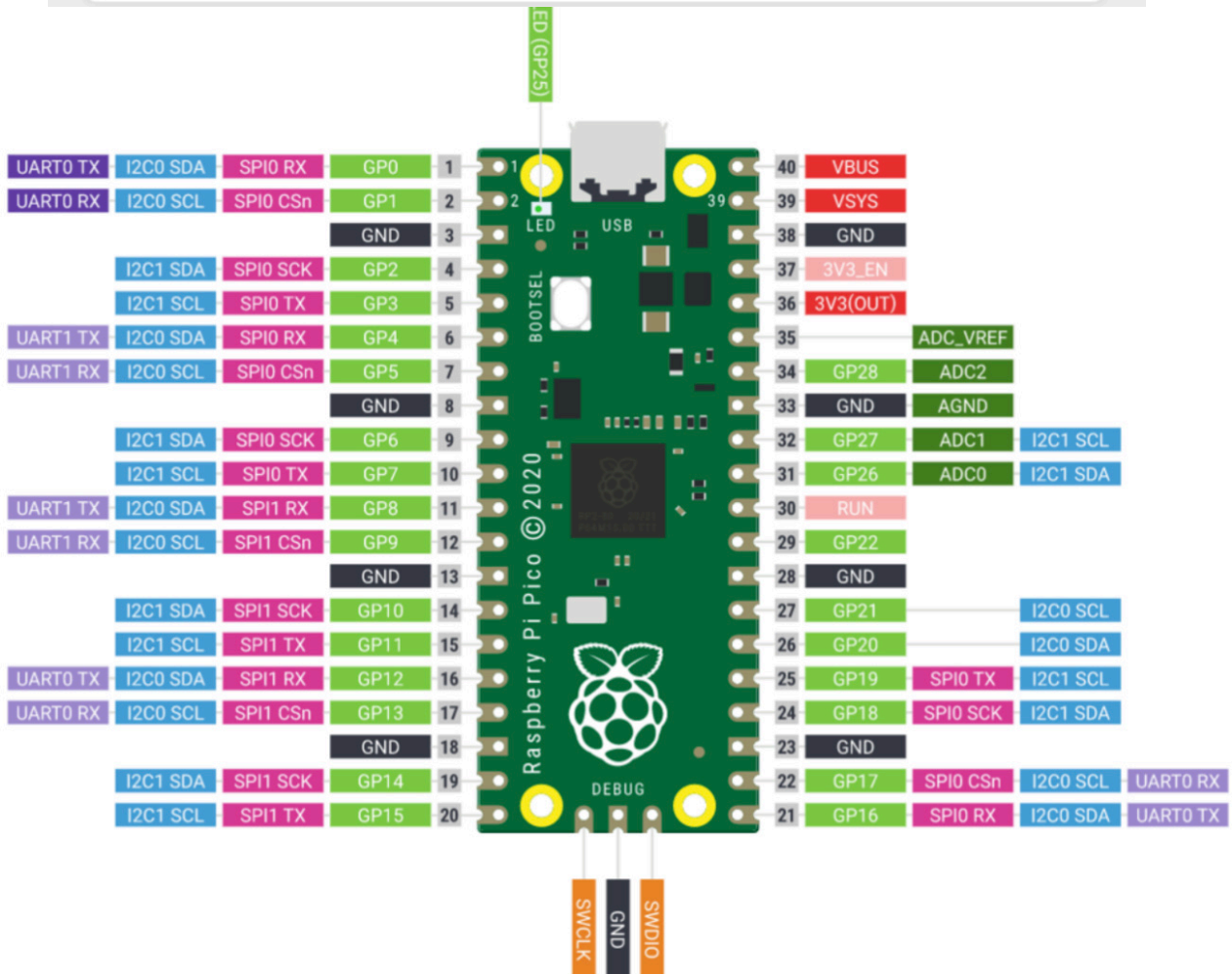
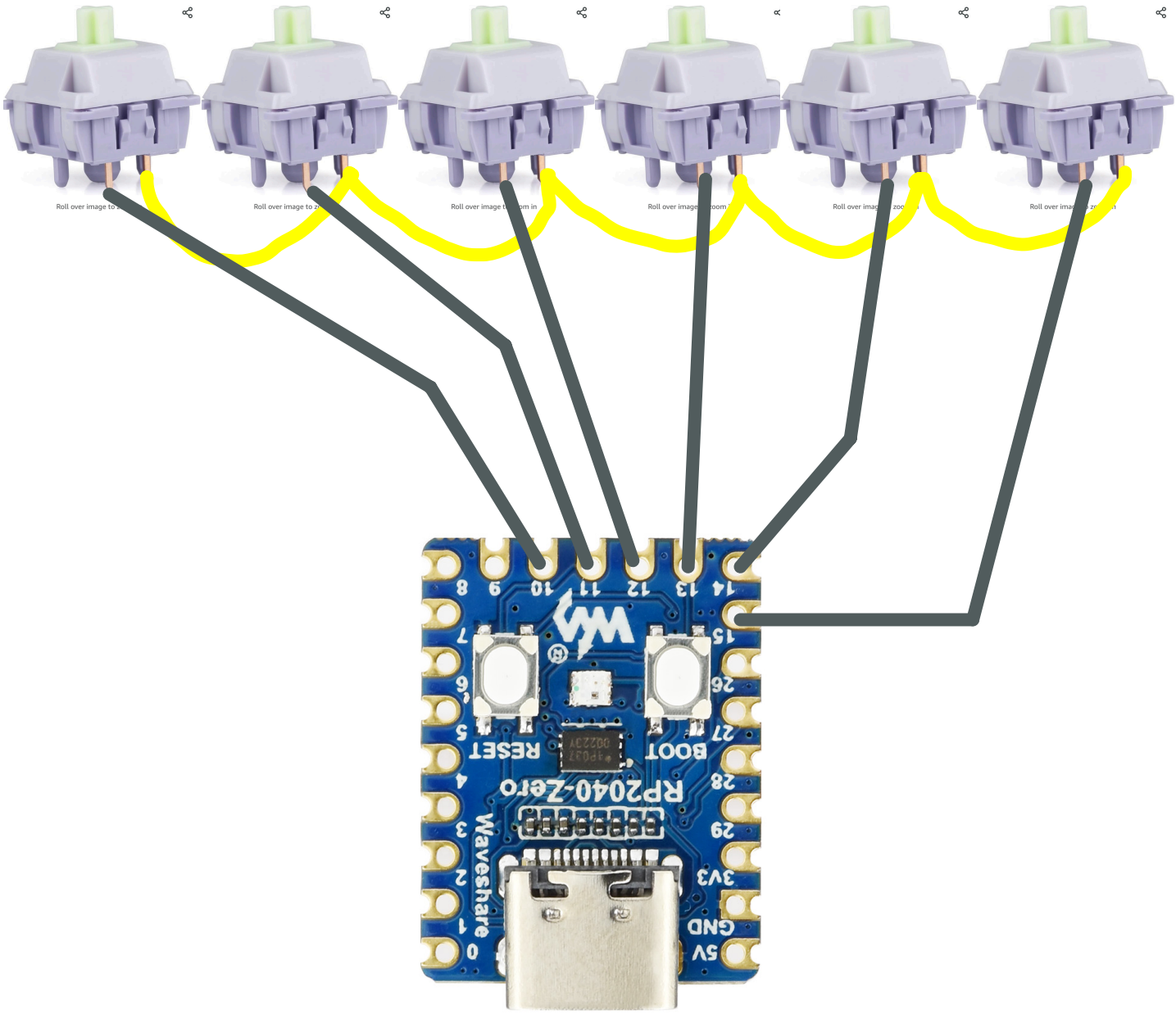
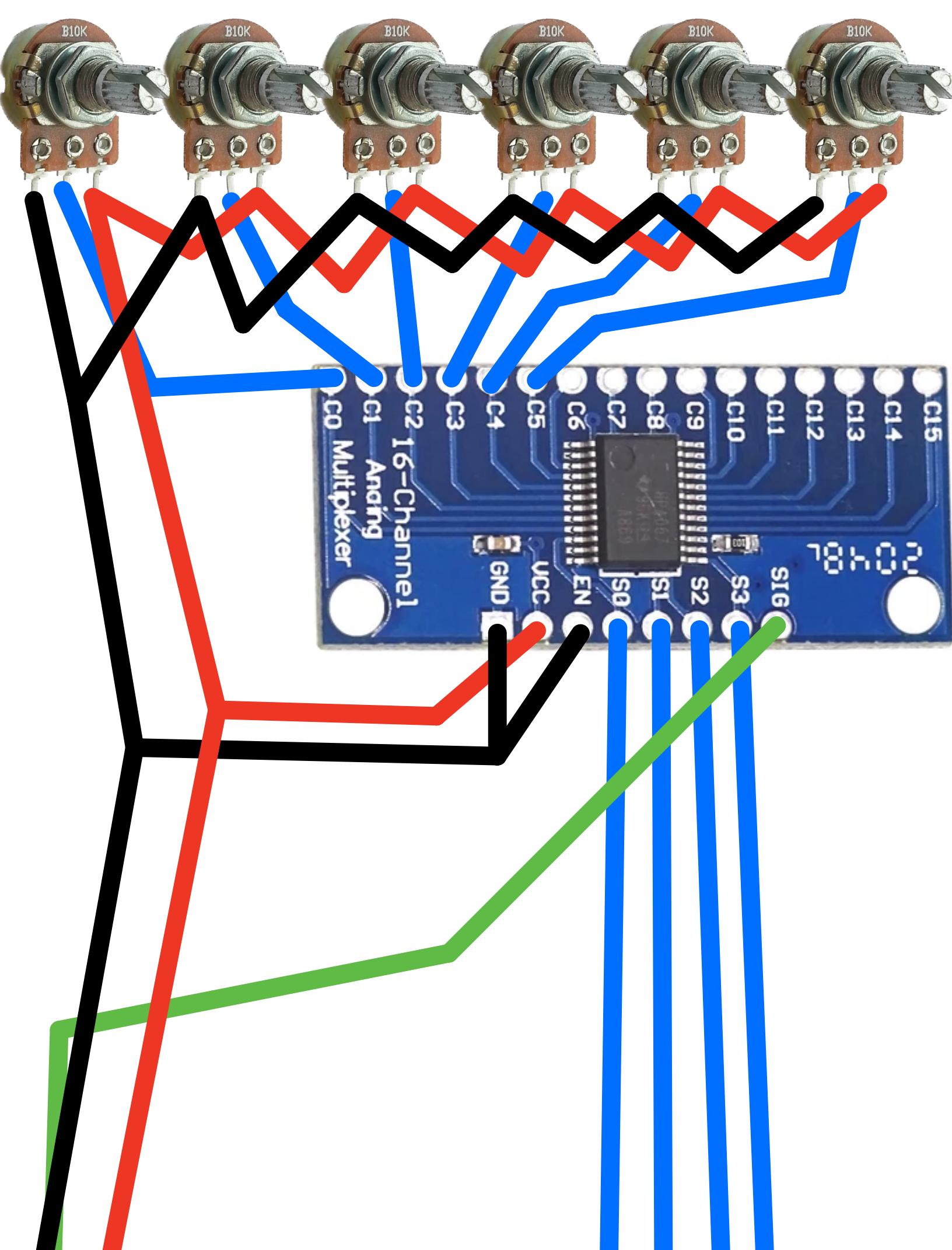


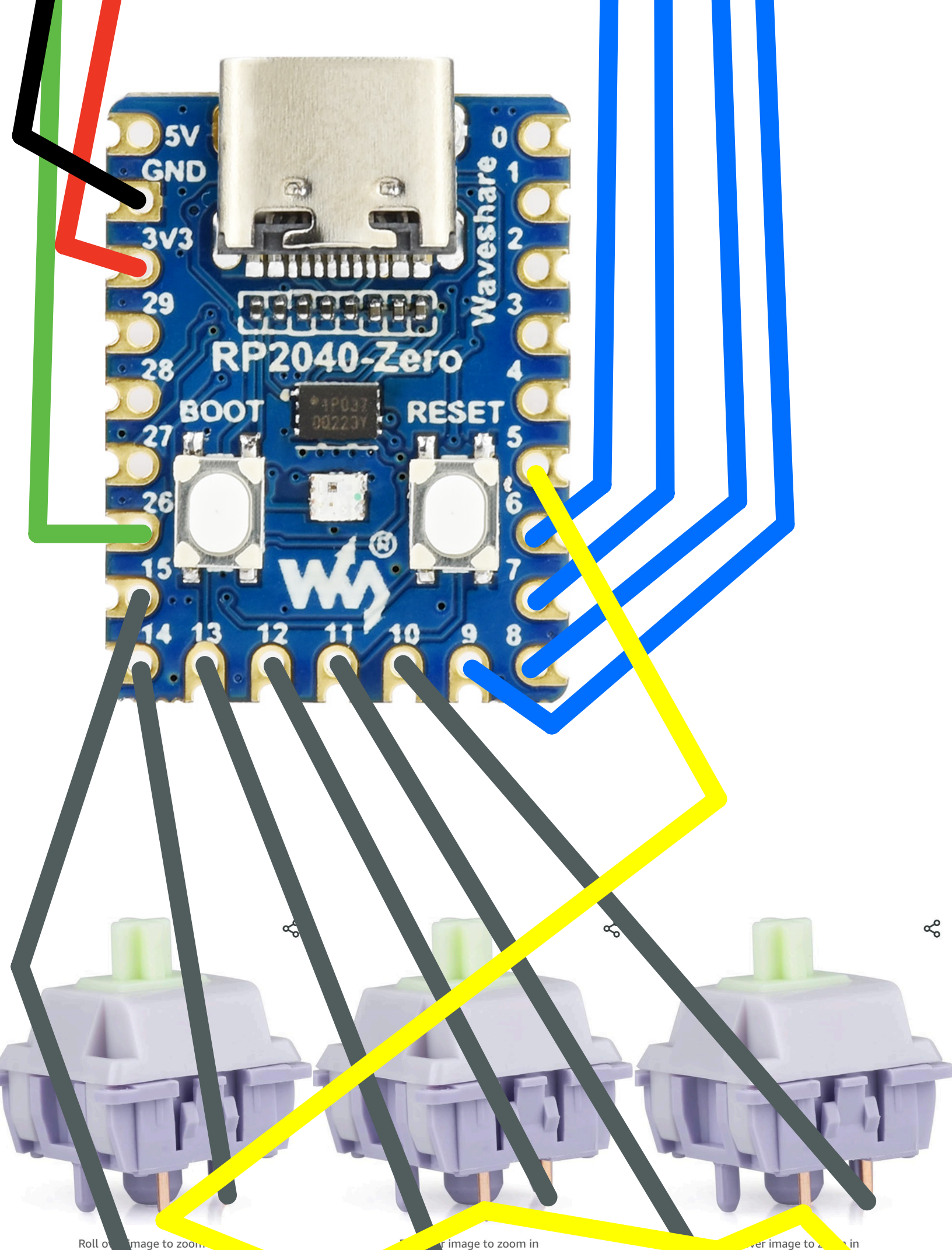
OK, I updated it and I hope, this time it should work


5V
GND
3V3
ADC3 GP29
ADC2 GP28
I2C1 SCL ADC1 GP27
I2C1 SDA ADC0 GP26
I2C1 SCL SPI1 TX GP15
I2C1 SDA SPI1 SCK GP14
RP2040-Zero
BOOT
RESET
Waveshare
GP0 SPI0 RX I2C0 SDA UART0 TX
GP1 SPI0 CSn I2C0 SCL UART0 RX
GP2 SPI0 SCK I2C1 SDA
GP3 SPI0 TX I2C1 SCL
GP4 SPI0 RX I2C0 SDA UART1 TX
GP5 SPI0 CSn I2C0 SCL UART1 RX
GP6 SPI0 SCK I2C1 SDA
GP7 SPI0 TX I2C1 SCL
GP8 SPI1 RX I2C0 SDA UART1 TX
GP9 SPI1 CSn I2C0 SCL UART1 RX
GP10 SPI1 SCK I2C1 SDA
GP11 SPI1 TX I2C1 SCL
GP12 SPI1 RX I2C0 SDA UART0 TX
GP13 SPI1 CSn I2C0 SCL UART0 RX
GND
GP25
GP24
GP23
GP22
GP21 I2C0 SCL
GP20 I2C0 SDA
GP19 SPI0 TX I2C1 SCL
GP18 SPI0 SCK I2C1 SDA
GP17 SPI0 CSn I2C0 SCL UART0 RX
WS2812 RGB LED used pin
GP16 DIN
Power
Ground
UART / UART (default)
GPIO, PIO, and PWM
ADC
SPI
Debugging
I2C


LED (GP25)
UART0 TX I2C0 SDA SPI0 RX GP0 1
UART0 RX I2C0 SCL SPI0 CSn GP1 2
GND 3
I2C1 SDA SPI0 SCK GP2 4
I2C1 SCL SPI0 TX GP3 5
UART1 TX I2C0 SDA SPI0 RX GP4 6
UART1 RX I2C0 SCL SPI0 CSn GP5 7
GND 8
I2C1 SDA SPI0 SCK GP6 9
I2C1 SCL SPI0 TX GP7 10
UART1 TX I2C0 SDA SPI1 RX GP8 11
UART1 RX I2C0 SCL SPI1 CSn GP9 12
GND 13
I2C1 SDA SPI1 SCK GP10 14
I2C1 SCL SPI1 TX GP11 15
UART0 TX I2C0 SDA SPI1 RX GP12 16
UART0 RX I2C0 SCL SPI1 CSn GP13 17
GND 18
I2C1 SDA SPI1 SCK GP14 19
I2C1 SCL SPI1 TX GP15 20
40 VBUS
39 VSYS
38 GND
37 3V3_EN
36 3V3(OUT)
35 ADC_VREF
34 GP28 ADC2
33 GND AGND
32 GP27 ADC1 I2C1 SCL
31 GP26 ADC0 I2C1 SDA
30 RUN
29 GP22
28 GND
27 GP21 I2C0 SCL
26 GP20 I2C0 SDA
25 GP19 SPI0 TX I2C1 SCL
24 GP18 SPI0 SCK I2C1 SDA
23 GND
22 GP17 SPI0 CSn I2C0 SCL UART0 RX
21 GP16 SPI0 RX I2C0 SDA UART0 TX
LED
USB
BOOTSEL
Raspberry Pi Pico ©2020
DEBUG
SWCLK
GND
SWDIO


5V
GND
3V3
29
28
27
26
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
BOOT
RESET
RP2040-Zero
Waveshare


B10K
B10K
B10K
B10K
B10K
B10K
C0
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
16-Channel
Analog
Multiplexer
GND
VCC
EN
S0
S1
S2
S3
SIG


5V
GND
3V3
29
28
27
26
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Waveshare
RP2040-Zero
BOOT
RESET

GND

3 volt Power

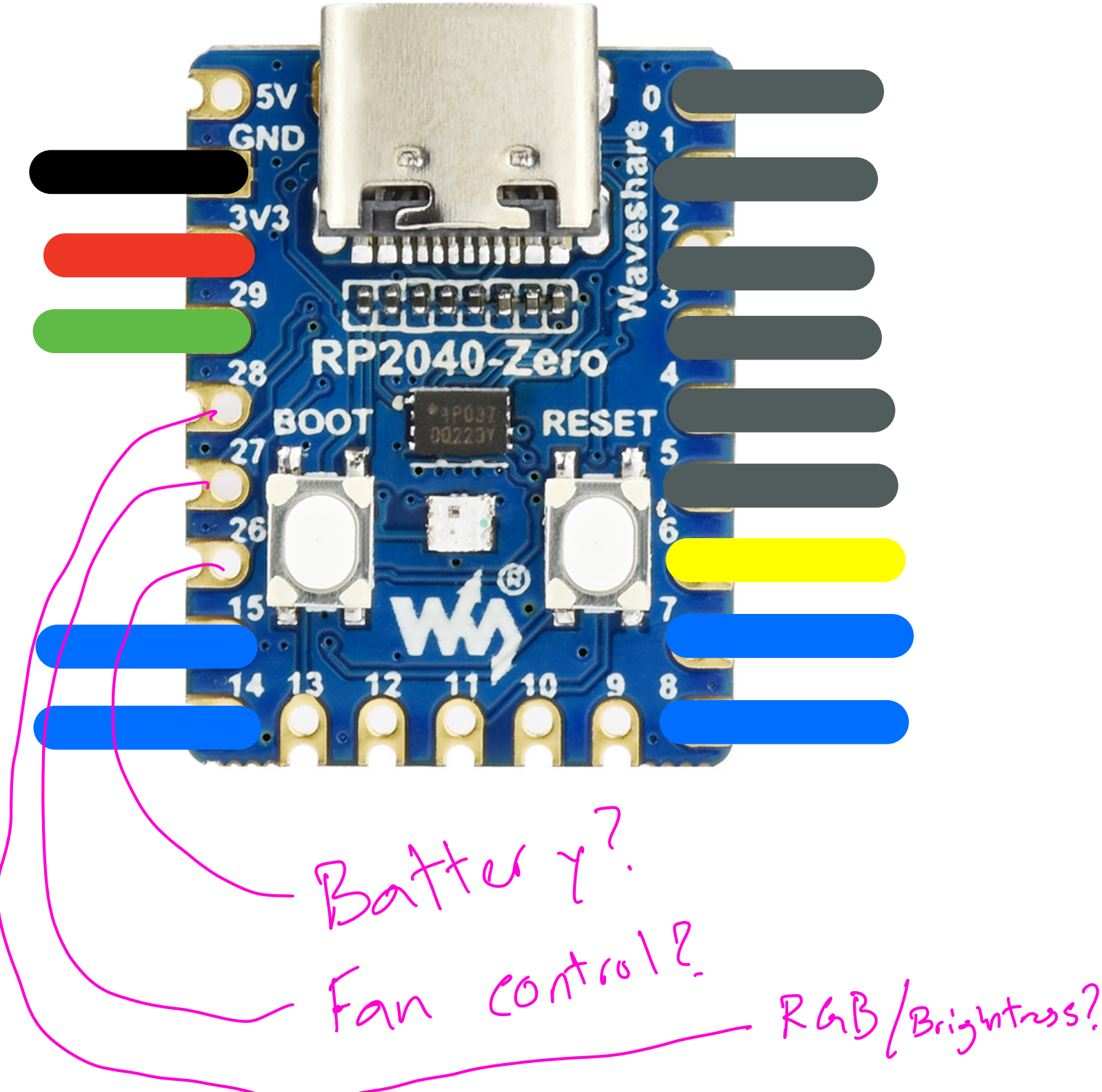
ADC signal

ADC Channel

Key Vertical

Key Horizontal

P.2.0 Wiring?


5V
GND
3V3
29
28
27
26
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Waveshare
RP2040-Zero
BOOT
RESET
Battery?
Fan control?
RGB/Brightness?

Absolutely, let's break down the code line by line for better understanding:

Imports (Lines 1-5):

```python
import board
from kmk.kmk_keyboard import KMKKeyboard
from kmk.keys import KC
from kmk.scanners import DiodeOrientation
from analogio import AnalogIn
from digitalio import Out  # For multiplexer selection
import yaml
import adafruit_hid  # For HID messages (optional)
```

1. `import board`: This line imports the `board` module which provides information about the specific microcontroller board you're using (in this case, WaveShare RP2040).
2. `from kmk.kmk_keyboard import KMKKeyboard`: This line imports the `KMKKeyboard` class from the `kmk.kmk_keyboard` module. This class is likely used to define your keyboard layout and functionality within the KMK framework.
3. `from kmk.keys import KC`: This line imports the `KC` class from the `kmk.keys` module. This class provides constants for various keyboard keys (e.g., `KC.A` for the "A" key).
4. `from kmk.scanners import DiodeOrientation`: This line imports the `DiodeOrientation` class from the `kmk.scanners` module. This class is likely used to specify the orientation of the diodes in your keyboard matrix (in this case, `DiodeOrientation.COL2ROW` for a matrix with diodes connecting columns to rows).
5. `from analogio import AnalogIn`: This line imports the `AnalogIn` class from the `analogio` module. This class is used to interact with analog input pins on your microcontroller, which will be used to read the potentiometer values.
6. `from digitalio import Out`: This line imports the `Out` class from the `digitalio` module. This class is used to control digital output pins, which are potentially needed for selecting channels on your multiplexer.
7. `import yaml`: This line imports the `yaml` module, which is used to read the configuration settings from a YAML file.
8. `import adafruit_hid` (optional): This line imports the `adafruit_hid` module. This might be used for future functionalities involving sending HID messages from your keyboard to control applications directly (not used in the current implementation relying on the deej library).

Board and Keyboard Setup (Lines 8-14):

```python
# Define microcontroller board
board = board.WaveShareRP2040

# Define row and column pin lists (same as before)
keyboard = KMKKeyboard()
keyboard.col_pins = (board.GP10, board.GP11, board.GP12, board.GP13, board.GP14, board.GP15)
keyboard.row_pins = (board.GP5,)
keyboard.diode_orientation = DiodeOrientation.COL2ROW
```

```

9. `board = board.WaveShareRP2040`: This line defines the microcontroller board you're using as `WaveShareRP2040`. This information is likely used by the KMK library for various functionalities.
10-12.  These lines define the keyboard object (`keyboard`) using the `KMKKeyboard` class. They specify the pins connected to the columns (`col_pins`) and the single row (`row_pins`) of your keyboard matrix. Finally, they set the `diode_orientation` to `DiodeOrientation.COL2ROW` based on your keyboard's diode configuration.

**Multiplexer Setup (Lines 15-20):**

```python
# Define potentiometer multiplexer selection pins (replace if needed)
mux_sel_pins = (board.GP6, board.GP7, board.GP8, board.GP9)
for pin in mux_sel_pins:
  # Initialize selection pins as outputs
  pin = Out(pin)
```

13. `mux_sel_pins = (board.GP6, board.GP7, board.GP8, board.GP9)`: This line defines the pins connected to the selection lines (S0-S3) of your multiplexer chip. These pins will be used to select specific channels on the multiplexer to read the potentiometer values. (Replace the pin numbers if your setup is different).
14-15. This loop iterates through each pin in the `mux_sel_pins` list and initializes them as outputs using the `Out` class from `digitalio`. This allows the code to control these pins and activate specific channels on the multiplexer.

**Multiplexer Selection Function (Lines 21-
```

```
import board
from kmk.kmk_keyboard import KMKKeyboard
from kmk.keys import KC
from kmk.scanners import DiodeOrientation
from analogio import AnalogIn
from digitalio import Out  # For multiplexer selection
import yaml
import adafruit_hid  # For HID messages (optional)

# Load configuration from YAML file (replace with your filename)
with open("config.yaml") as f:
  config = yaml.safe_load(f)

# Define microcontroller board
board = board.WaveShareRP2040

# Define row and column pin lists (same as before)
keyboard = KMKKeyboard()
keyboard.col_pins = (board.GP10, board.GP11, board.GP12, board.GP13, board.GP14, board.GP15)
keyboard.row_pins = (board.GP5,)
keyboard.diode_orientation = DiodeOrientation.COL2ROW

# Define potentiometer multiplexer selection pins (replace if needed)
mux_sel_pins = (board.GP6, board.GP7, board.GP8, board.GP9)
for pin in mux_sel_pins:
  # Initialize selection pins as outputs
  pin = Out(pin)

# Define function to select a channel on the multiplexer
def select_mux_channel(channel):
  # Set selection lines (S0-S3) based on the channel number (adjust logic if needed)
  for i in range(4):
    mux_sel_pins[i].value = (channel >> i) & 1  # Bitwise operation for selection

# Define potentiometer pin (through multiplexer, SIG connected to GP26)
potentiometer_pin = board.GP26

# Define keymap (same as before)
keyboard.keymap = [[KC.A, KC.B, KC.C, KC.D, KC.E, KC.F]]

# Initialize deej object (replace with your COM port and baud rate)
com_port = config["com_port"]
baud_rate = config["baud_rate"]
deej_obj = deej.Deej(com_port, baud_rate)  # Assuming deej library is installed

# Main loop
if __name__ == '__main__':
```

```
  potentiometer = AnalogIn(potentiometer_pin)  # Initialize potentiometer object
  while True:
    # Loop through each multiplexer channel (adjust number if needed)
    for channel in range(4):
      select_mux_channel(channel)  # Select the current channel
      pot_value = potentiometer.value  # Read potentiometer value

      # Normalize potentiometer value to 0-1 range
      normalized_value = pot_value / (2**16 - 1)  # Assuming 16-bit ADC

      # Get slider mapping and invert setting from configuration
      slider_mapping = config["slider_mapping"]
      invert_sliders = config["invert_sliders"]

      # Invert value if needed
      if invert_sliders:
        normalized_value = 1 - normalized_value

      # Control volume based on potentiometer position and mapping
      slider_index = int(normalized_value * len(slider_mapping))  # Select slider based on normalized
value
      application = slider_mapping.get(slider_index)  # Get application name from mapping

      if application:
        # Use deej object to control volume for the specific application
        deej_obj.set_volume(application, normalized_value)  # Assuming deej supports setting specific
app volume
      else:
        # Handle cases where no application is mapped to the slider position (optional)
        pass

    # Rest of your keyboard code (e.g., keymap processing)
    keyboard.tick()  # Call keyboard tick function (likely needed for KMK)
```

Python

```python
import board
from kmk.kmk_keyboard import KMKKeyboard
from kmk.keys import KC
from kmk.scanners import DiodeOrientation
from analogio import AnalogIn
from digitalio import Out  # For multiplexer selectic
import yaml
import adafruit_hid  # For HID messages (optional)

# Load configuration from YAML file (replace with you
with open("config.yaml") as f:
  config = yaml.safe_load(f)

# Define microcontroller board
board = board.WaveShareRP2040

# Define row and column pin lists (same as before)
keyboard = KMKKeyboard()
keyboard.col_pins = (board.GP10, board.GP11, board.GP
keyboard.row_pins = (board.GP5,)
keyboard.diode_orientation = DiodeOrientation.COL2ROW

# Define potentiometer multiplexer selection pins (re
mux_sel_pins = (board.GP6, board.GP7, board.GP8, boar
for pin in mux_sel_pins:
  # Initialize selection pins as outputs
  pin = Out(pin)

# Define function to select a channel on the multiple
def select_mux_channel(channel):
  # Set selection lines (S0-S3) based on the channel
  for i in range(4):
    mux_sel_pins[i].value = (channel >> i) & 1  # Bit

# Define potentiometer pin (through multiplexer, SIG
potentiometer_pin = board.GP26

# Define keymap (same as before)
keyboard.keymap = [[KC.A, KC.B, KC.C, KC.D, KC.E, KC.

# Initialize deej object (replace with your COM port
com_port = config["com_port"]
baud_rate = config["baud_rate"]
deej_obj = deej.Deej(com_port, baud_rate)  # Assuming

# Main loop
if __name__ == '__main__':
  potentiometer = AnalogIn(potentiometer_pin)  # Init
  while True:
    # Loop through each multiplexer channel (adjust r
    for channel in range(4):
      select_mux_channel(channel)  # Select the curre
      pot_value = potentiometer.value  # Read potenti

      # Normalize potentiometer value to 0-1 range
      normalized_value = pot_value / (2**16 - 1)  # A

      # Get slider mapping and invert setting from co
      slider_mapping = config["slider_mapping"]
      invert_sliders = config["invert_sliders"]

      # Invert value if needed
      if invert_sliders:
        normalized_value = 1 - normalized_value

      # Control volume based on potentiometer positic
      slider_index = int(normalized_value * len(slide
      application = slider_mapping.get(slider_index)

      if application:
        # Use deej object to control volume for the s
        deej_obj.set_volume(application, normalized_v
      else:
        # Handle cases where no application is mappec
        pass

    # Rest of your keyboard code (e.g., keymap proces
    keyboard.tick()  # Call keyboard tick function (l
```